Bridge Design Pattern
Members: Daugdaug, Justine; Pandiyan, Janessa; Salera, Lance Vincent

Definition

The Bridge Design Pattern divides and organizes a single class that has multiple variants of some functionality into two hierarchies: abstractions and implementations. By doing this, the client code won't be exposed to implementation details as it will only work with high level abstractions.

History

The bridge design pattern is one of the 23 design patterns introduced by the Gang of Four, composed of Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, on a book called "Design Patterns: Elements of Reusable Object-Oriented Software", published on October 21, 1994.

Usages

- Decoupling Interface and Implementation
    - Separate high-level logic from low-level operations.
    - Independent evolution of both parts.
- Platform Independence
    - Operate across multiple platforms/environments.
    - Example: Different graphics rendering engines (DirectX, OpenGL).

- Extensibility
    - Extend main logic and operations without mutual impact.
    - Add features without altering core functionality.
- Dynamic Binding
    - Determine implementation at runtime.
    - Useful for plugin architectures and dynamic module loading.

Real-world examples

- The Bridge pattern allows e-commerce platforms to integrate multiple payment gateways without changing the checkout interface.
- The Bridge Design Pattern is applicable to music player apps that handle many file formats. The Bridge pattern is used to separate the abstraction (the music player interface) from the implementation (file format handling) in this classic example.

Pros

- Cleaner Code
    - Separation of concerns leads to more organized and readable code.
- Maintainability
    - Changes in implementation don't affect the client code.
- Scalability
    - Easily introduce new implementations without major code
- Flexibility
    - Mix and match different implementations as needed.

Cons

- Complexity
    - When abstractions and implementations increase, the Bridge pattern might complicate a program. Code complexity can make it difficult to comprehend and maintain.

- Multiple Indirections

    - Several indirections within an application has the potential to adversely impact its performance. This can be attributed to the inherent indirection that occurs when a request is passed from the Abstraction to the Implementor.